

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

**APPLICATION FOR LETTERS PATENT**

**INVALID POLICY DETECTION**

Inventor(s):  
Jeffrey C. Schlimmer  
David Levin  
Alfred Lee  
Erik B. Christensen  
Bradford H. Lovering

ATTORNEY'S DOCKET NO. MS1-1853US  
CLIENT'S DOCKET NO. 306881.01

## **INVALID POLICY DETECTION**

### **RELATED APPLICATIONS**

[0001] This patent application is related to co-owned U.S. Patent Application No. \_\_\_\_\_ (MS306882.01/MS1-1855US) and U.S. Patent Application No. \_\_\_\_\_ (MS306888.01/MS1-1861US), each hereby incorporated herein for all that is disclosed.

### **TECHNICAL FIELD**

[0002] The described subject matter relates to electronic computing, and more particularly to systems and methods of invalid policy detection in electronic computing systems.

### **BACKGROUND**

[0003] Communication between various computing devices (e.g., personal computers, server computers, mobile devices) is increasingly commonplace in a number of network environments, such as, e.g., the Internet and corporate intranets to name only a few examples. Often, these computing devices are configured for communication in accordance with preferred or even required formats. As an illustration, a commercial web site may require a user's computer to comply with one or more assertions before the user is granted access to the payment web pages.

1 These assertions may be specified in one or more policies and may be either  
2 general or security related. For example, an assertion may require that incoming  
3 messages be encoded according to a particular encryption scheme for security  
4 purposes, or that incoming messages be formatted using a particular compression  
5 scheme to facilitate efficient transaction processing.

6 [0004] The user's computer, or client, may request a copy of a policy from a  
7 host (e.g., a server at the commercial website). The client reads the assertions  
8 included in the policy and complies with one or more of the assertions to  
9 communicate with or access resources via the host. The client may also store a  
10 copy of the policy in a cache so that the client does not have to request another  
11 copy of the policy the next time the client communicates with the host. However,  
12 policies may be changed (e.g., to implement updated encryption schemes).  
13 Accordingly, a client that is using an "old" policy from cache may no longer be  
14 complying with valid assertions when attempting to communicate with or access  
15 resources via the host.  
16  
17

18 [0005] If the client is not complying with valid assertions, the host may  
19 simply ignore messages from the client. Optionally, the host may issue an error  
20 message. However, these error messages are also used to indicate other types of  
21 errors. For example, an "invalid action" message may also be issued in response to  
22 the client attempting to access a resource that does not exist. An "invalid security"  
23 message may also be issued in response to the client submitting improper security  
24  
25

1 credentials. These messages do not indicate to the client that the cached policy is  
2 invalid, and the client may continue to use the cached policy, albeit unsuccessfully.

### 3 4 SUMMARY

5 [0006] Implementations are described and claimed herein to detect an  
6 invalid policy that may reside in a cache at a client. According to one  
7 implementation, a policy is removed from cache if it expires. If the cached policy  
8 is expired, a current policy is requested. Otherwise the cached policy may be used  
9 to communicate with a host. The client indicates which policy it is using by  
10 generating a policy digest, including, in compressed form, the policy identity and  
11 one or more assertions that the client is complying with. The client sends the policy  
12 digest to the host, e.g., as part of a message header.

13  
14 [0007] If the host determines the policy identity is invalid, the assertions do  
15 not need to be evaluated and the host issues an invalid digest fault. If the policy is  
16 valid, but the assertions indicated by the policy digest are invalid, the host issues  
17 an invalid policy fault. In either case, the client is notified that the cached policy is  
18 no longer valid and that a current policy should be requested.

19  
20 [0008] In some implementations, articles of manufacture are provided as  
21 computer program products. One implementation of a computer program product  
22 provides a computer program storage medium readable by a computer system and  
23 encoding a computer program for invalid policy detection. Another implementation  
24  
25

1 of a computer program product may be provided in a computer data signal  
2 embodied in a carrier wave by a computing system and encoding the computer  
3 program for invalid policy detection.

4 [0009] The computer program product encodes a computer program for  
5 executing a computer process on a computer system to generate a policy digest for  
6 a cached policy at a client. The policy digest identifies the policy and at least one  
7 assertion that the client is complying with. The policy digest is included in a  
8 request by the client to access a resource.

10 [0010] In another implementation of the computer program product, a  
11 computer process extracts at a host a policy digest identifying a cached policy from  
12 a request to access a resource; and grants access to the resource if the policy digest  
13 identifies a valid policy.

14 [0011] In yet another implementation, a method is provided. The method  
15 includes generating a policy digest for a cached policy at a client, the policy digest  
16 identifying the policy and at least one assertion that the client is complying with.  
17 The policy digest is included in a request by the client to access a resource.

19 [0012] In another implementation, a method includes extracting at a host a  
20 policy digest identifying a cached policy from a request to access a resource.  
21 Access is granted to the resource if the policy digest identifies a valid policy (and  
22 the host verifies that the request complies with the indicated assertions).  
23  
24  
25

1 [0013] In yet another implementation, a system is provided including a  
2 policy digest identifying at least one cached policy. A messaging module grants  
3 access to a resource if the policy digest identifies a valid policy for the resource  
4 (and the host verifies that the request complies with the indicated assertions).

5 [0014] In another implementation of the system, a policy digest identifies  
6 the policy and at least one assertion that the client is complying with. A messaging  
7 module includes the policy digest in a request by the client to access a resource.  
8

### 9 **BRIEF DESCRIPTION OF THE DRAWINGS**

10  
11 [0015] Fig. 1 is a schematic illustration of an exemplary computer network  
12 that may implement policy-based communication;

13 [0016] Fig. 2 is a schematic diagram of an exemplary client and host  
14 implemented to detect an invalid policy;

15 [0017] Fig. 3 is a high-level diagram illustrating an exemplary  
16 implementation of invalid policy detection;  
17

18 [0018] Figs. 4-6 are flowcharts illustrating exemplary operations to  
19 implement invalid policy detection; and

20 [0019] Fig. 7 is a schematic illustration of an exemplary computing device  
21 that can be utilized to implement invalid policy detection.  
22  
23  
24  
25

## DETAILED DESCRIPTION

1 [0020] Briefly, invalid policy detection may be implemented for a client  
2 attempting to access a host resource (or otherwise communicate with a host  
3 computer) in a network environment, such as, e.g., the Internet or an intranet. For  
4 purposes of illustration, a user may attempt to access a secure payment web page  
5 to make a purchase at a commercial web site. Before the user is able connect to the  
6 secure payment web page, however, the host has to be assured that the client is  
7 complying with at least one assertion included in a policy associated with the  
8 resource. For example, the assertions may require that messages sent by the client  
9 comply with a specified encryption scheme, compression scheme, or passport  
10 signature.  
11

12  
13 [0021] In operation, the client retrieves a copy of the policy from cache and  
14 determines whether the cached policy is current or expired. If the cached policy is  
15 expired, the client requests a current policy from the host. If the cached policy is  
16 current, the client can use the cached policy to access, e.g., the secure payment  
17 website, by complying with one or more of the assertions included in the cached  
18 policy. The client identifies the cached policy and indicates which assertions the  
19 client is complying with by generating a policy digest and sending the policy digest  
20 to the host to validate.  
21

22  
23 [0022] The host receives the policy digest from the client (e.g., as part of a  
24 message header) and determines if the client is complying with a valid policy. If  
25

1 the policy identified by the policy digest is invalid, the host returns a fault  
2 notifying the client that the cached policy is no longer valid and a current policy  
3 will need to be requested before the secure payment web page (or other host  
4 resource) can be accessed.

5 [0023] Accordingly, the client receives informative feedback and can readily  
6 determine whether the cached policy is valid or should be removed from cache. In  
7 addition, the host is able to quickly process messages from a client by evaluating a  
8 compressed policy digest indicating the particular assertions that the client is  
9 complying with. The host only has to validate assertions indicated in the policy  
10 digest and does not have to test every potential assertion. The declarative nature of  
11 the policy digest also mitigates against attackers who may insert destructive tokens  
12 in the message at intermediary points in the network.  
13  
14  
15

### 16 **Exemplary System**

17 [0024] Fig. 1 is a schematic illustration of an exemplary networked  
18 computing system 100 in which invalid policy detection may be implemented. The  
19 networked computer system 100 may include one or more communication  
20 networks, such as local area network (LAN) 110 and/or wide area network (WAN)  
21 120. One or more hosts 130a, 130b and client devices 140a-f may be linked over  
22 the communication network(s) 110, 120.  
23  
24  
25



1 [0025] Hosts 130a, 130b and clients 140a-f connect to a network via a  
2 communication connection such as, e.g., an Ethernet connection. Alternatively, a  
3 host and/or client can connect directly, as illustrated by direct connection 150.  
4 Although there are no theoretical limits on the number of hosts and clients that can  
5 be included in a network such as computer network 100, the number of hosts and  
6 clients are limited primarily by the connectivity implemented in the  
7 communication network.

8 [0026] The terms "client" and "host" both refer to the hardware and  
9 software (the entire computer system) used to perform various computing services.  
10 The computer system may be implemented as a server computer that is dedicated  
11 to server applications or that also runs other applications. Alternatively, the  
12 computer system may include, by way of illustration, stand-alone personal desktop  
13 or laptop computers (PCs), workstations, personal digital assistants (PDAs), or  
14 electronic appliances, to name only a few examples.

15 [0027] As used herein, the client is the computer system that is used to  
16 access a resource at another computer system (i.e., the host). Resources available  
17 via the host may include other computing or data processing systems, storage, or  
18 other devices. The host may also provide the client with other services, such as  
19 transaction processing, email services, etc. A secured host can also determine  
20 whether the client is allowed access to private resources, provide authorized users  
21  
22  
23  
24  
25

1 access to private resources, and generate audit messages (e.g., in an event log)  
2 when the client attempts to access private resources.

3 [0028] Although the host may be implemented as a server computer and a  
4 client may be implemented as, e.g., a personal computer or workstation, any  
5 computer system can serve as a client and/or as a host. As an illustration, if a  
6 network workstation is accessing a resource via a front-end server, the network  
7 workstation is referred to herein as the client and the front-end server is referred to  
8 herein as the host. However, if the front-end server is accessing a resource via a  
9 back-end server, the front-end server is referred to herein as the client and the  
10 back-end server is referred to herein as the host. As another illustration, if a first  
11 PC is accessing a resource via a second PC (e.g., peer-to-peer), the first PC is  
12 referred to herein as the client and the second PC is referred to herein as the host.

14 [0029] A client may communicate with a host according to one or more  
15 policies 160, 165 bound to a policy subject, such as, e.g., a resource at the host.  
16 Generally, a policy is an informal abstraction of information referred to as  
17 assertions. An assertion represents an individual preference, requirement,  
18 capability, or other property that a client may, or in some circumstances, must  
19 comply with in order to access the resource via (or otherwise communicate with)  
20 the host. Assertions may be either general or security-related.

22 [0030] In an exemplary implementation, the Web Services Policy (WS-  
23 Policy) specification defines a general model and syntax for policy expressions and  
24

1 assertions. The policy may be expressed in machine-readable extensible markup  
2 language (XML) format to facilitate interoperability between different platforms  
3 and web services infrastructure. Of course a policy is not limited to any particular  
4 syntax or format and other implementations are also possible.

5 [0031] During operation, a client may request a copy of one or more policies  
6 from a host. The client may store the policy locally, e.g., in cache. The client is  
7 then able to access the policy from cache for other interactions with the host and  
8 does not have to request another copy of the policy each time the client interacts  
9 with the host. If the cached policy becomes invalid, the client removes the policy  
10 from cache and requests a valid policy from the host.

12 [0032] Fig. 2 is a schematic diagram of an exemplary client and host that  
13 may be implemented to detect an invalid policy. Client 200 connects to a host 230  
14 via a network 260 or direct connection (e.g., as illustrated in Fig. 1 by connection  
15 150). In Fig. 2, client 200 connects to the network 260 via client port 201 and host  
16 230 connects to the network 260 via host port 231.

18 [0033] Referring to Fig. 2, client 200 includes a client messaging module  
19 210 to process messages. Client messaging module 210 is operatively associated  
20 with a cache 220 including one or more cached policies 221a, 221b, 221c  
21 (hereinafter generally referred to as 221). The client 200 may comply with  
22 assertion(s) included in one or more of the cached policies 221 to communicate  
23 with the host 230.  
24  
25

1 [0034] In an exemplary implementation, client messaging module 210 may  
2 communicate with the host 230 using Simple Object Access Protocol (SOAP).  
3 SOAP is a messaging protocol used to encode transactions for transfer over a  
4 network using any of a variety of Internet protocols (e.g., HTTP, SMTP, MIME).  
5 SOAP messages do not need to be formatted for use with any particular operating  
6 system, making SOAP messages commonplace in network environments.

7 [0035] Client messaging module 210 may be implemented in computer-  
8 readable program code (e.g., software and/or firmware) stored in computer-  
9 readable storage or memory and executable by a processor (or processing units) at  
10 the client 200. Client messaging module 210 may include a digest generator 212, a  
11 message generator 214, and a fault processor 216.

13 [0036] Digest generator 212 may be implemented to generate a policy digest  
14 of one or more policies stored in cache 220. Techniques to generate a policy digest  
15 are described in more detail below with reference to Fig. 3. For now it is sufficient  
16 to understand that the policy digest includes the identity of at least one policy and  
17 at least one assertion in a compressed format. Accordingly, the policy digest can be  
18 added to a message (e.g., as part of the header) and sent to the host 230 without  
19 having to include a copy of the entire policy.  
20

21 [0037] Message generator 214 may be implemented to generate messages at  
22 the client. Message generator 214 may also include a copy of the policy digest as  
23 part of the message (e.g., in the message header) to identify the policy and  
24  
25

1 assertion(s) that the client is complying with. For purposes of illustration, a  
2 message may include a request to access a resource at host 230. The policy digest  
3 may identify one or more policies and assertions that client 200 is complying with  
4 so that the host 230 will grant the client 200 access to the resource.

5 [0038] Fault processor 216 may be implemented to process faults received  
6 by client 200 in response to a message that the client sent. For example, if the host  
7 230 determines that client 200 is not complying with a valid policy, the host 230  
8 may issue a fault. Fault processor 216 reads the fault and determines which policy  
9 is invalid so that the client messaging module 210 can remove the invalid policy  
10 from cache 220 and request a valid policy before attempting to access the resource  
11 at host 230.  
12

13 [0039] Still referring to Fig. 2, host 230 includes a host messaging module  
14 240 to process messages at the host. Host messaging module 240 is operatively  
15 associated with computer-readable storage or memory 250 including one or more  
16 policies 251a, 251b, 251c (hereinafter generally referred to as 251). Host  
17 messaging module 240 may request, or in some circumstances, require that a client  
18 200 comply with one or more of the policies 251.  
19

20 [0040] Host messaging module 240 may be implemented in computer-  
21 readable program code (e.g., software and/or firmware) stored in computer-  
22 readable storage or memory and executable by a processor (or processing units) at  
23  
24  
25

1 the host 230. Host messaging module 240 may include a message processor 242, a  
2 digest processor 244, and a fault generator 246.

3 [0041] Message processor 242 may be implemented to process messages at  
4 the host 230. For purposes of illustration, the message may include a request to  
5 access a resource at host 230. The message processor 242 extracts the policy  
6 digest, which is processed by the digest processor 244 to determine whether the  
7 client is complying with a valid policy. Techniques to extract a policy digest and  
8 determine whether the client is complying with a valid policy are described in  
9 more detail below with reference to Fig. 3. If the host is complying with a valid  
10 policy, the message processor 242 may process the message to grant the client 200  
11 access to the resource at the host 230.  
12

13 [0042] Fault generator 246 may be implemented to generate faults when a  
14 client is not complying with a valid policy. Fault processor 216 generates a fault  
15 that can be returned to the client 200 so that the client 200 can remove the invalid  
16 policy from cache 220 and request a valid policy before attempting to access the  
17 resource at host 230. The fault may trigger a diagnostic event at the client 200 to  
18 indicate a system problem. For example, the diagnostic event may be or displayed  
19 for the user or logged for a system administrator's review.  
20

21 [0043] It is noted that the client messaging module 210 and the host  
22 messaging module 240 are not limited to the exemplary implementation shown in  
23 Fig. 2. For example, the functions do not need to be embodied in separate modules.  
24  
25

1 In yet other implementations, additional functional components may also be  
2 included.

3 [0044] Client 200 may also include a routing module 270 operatively  
4 associated with a routing table 271. Routing table 271 defines one or more network  
5 paths to route messages to the host. Routing module 270 may be implemented as  
6 computer-readable program code (e.g., software and/or firmware) stored in  
7 computer-readable storage or memory and executable by a processor (or  
8 processing units) at the client 200.

9  
10 [0045] Routing module 270 may be operatively associated with the fault  
11 processor 216. When the fault processor 216 receives a fault that indicates a host is  
12 no longer available, fault processor 216 notifies routing module 270 to update the  
13 routing table 271 by removing the unavailable host from the routing table 271.

14 [0046] Fig. 3 is a high-level diagram illustrating an exemplary  
15 implementation of invalid policy detection. A cached policy 310 may include one  
16 or more assertions 315. For purposes of illustration, exemplary cached policy 310  
17 includes assertions (A1 or A2) and (A3 or A4).

18  
19 [0047] In operation, a client 300 may have to comply with one or more  
20 policies to access a resource at a host 350. The client 300 determines if it has  
21 previously received a policy from the host 350 and stored it in cache. The client  
22 300 accesses the cached policy 310 and selects assertions to comply with.  
23  
24  
25

1 [0048] The client 300 may generate a policy digest 320 to identify the policy  
2 (or policies) and selected assertion(s). In an exemplary implementation, policy  
3 digest 320 is a canonicalized XML document. Policy digest 320 may be generated,  
4 for example, by computing a policy identity, and identifying selected assertions  
5 within that policy (or policies).

6 [0049] The policy identity may be computed by hashing the text  
7 representation of the policy. For example, a cryptographic hash function (e.g.,  
8 MD5) may be used to compute the policy identity as a binary value. MD5 is an  
9 algorithm that takes input of arbitrary length and produces a 128 bit “fingerprint”  
10 of the input. The MD5 algorithm is commonly used for digital signature  
11 applications where a large file must be compressed in a secure manner before  
12 being encrypted with a private key under a public key implementation. Of course  
13 other implementations now known or later developed may also be used.

14 [0050] The binary value may be encoded as a text value (e.g., using base64  
15 encoding), which is included in the policy digest in XML format. Base64 encoding  
16 is specified in the Network Working Group’s Request for Comments (RFC 2045 –  
17 Multipurpose Internet Mail Extensions). Base64 encoding uses a 64 character  
18 subset (A-Za-z0-9+/) to represent binary data and = for padding. This encoding  
19 processes data as 24 bit groups and maps the data to four encoded characters.  
20 Every 6 bits of the 24 bit group serves as an index to a mapping table to obtain a  
21  
22  
23  
24  
25



1 character for the encoded data. Of course other implementations now known or  
2 later developed may also be used.

3 [0051] The selected assertions may be identified using a bit vector, wherein  
4 the bit vector includes one bit for each assertion in the policy, and a bit is set (e.g.,  
5 the bit value is one) only if the corresponding assertion is selected. The binary  
6 value may be encoded as a text value (e.g., base64) for inclusion in the policy  
7 digest in XML format.

8 [0052] For purposes of illustration, a bit vector 330 may be generated by  
9 iterating through each assertion 315 in the policy 310, and in parallel, iterating  
10 through each bit 331a-d in the bit vector 330. For example, the first assertion A1  
11 may correspond to the first bit 331a, the second assertion A2 may correspond to  
12 the second bit 331b, and so forth. Bit values are assigned to each assertion to  
13 indicate whether the assertion is selected to satisfy the policy. The corresponding  
14 bit is assigned a bit value of "1" if the assertion is selected and a bit value of "0" if  
15 the assertion is not selected. The resulting bit vector 330 represents the policy  
16 digest 320.

17 [0053] The policy digest may be included in a message 340, e.g., as part of  
18 the header 345. When message 340 is received by host 350, the policy digest is  
19 extracted from the message 340 and decoded (e.g., using a table) to reconstruct the  
20 policy digest 320 at the host 350. The policy digest 320 is then compared to the  
21 host policy 360. If the policy digest 320 indicates that the client is complying with  
22  
23  
24  
25

1 a valid policy, the client 300 may be granted access to the requested resource.

2 Alternatively, if the policy digest 320 indicates that the cached policy 310 is  
3 invalid, the host 350 issues a fault 370. Fault 370 may indicate to the client 300  
4 that the cached policy 310 is invalid.

5 [0054] In another illustrative implementation, a policy digest may be  
6 generated as follows. The policy assertions are first placed in canonical  
7 (alphabetical) order:

8 <Policy>  
9 <FirstAssertion/>  
10 <SecondAssertion/>  
11 <SecurityAssertion>  
12 <Feature1/>  
13 <Feature2>  
14 <Annotation> first stuff for service </Annotation>  
15 </Feature2>  
16 <Feature3/>  
17 <Annotation> second stuff </Annotation>  
18 </SecurityAssertion>  
19 <XAlmostLastAssertion>  
20 <FeatureA>  
21 <FeatureB>  
22 <Annotation> stuff for AlmostLast </Annotation>  
23 </XAlmostLastAssertion>  
24 <XLastAssertion/>  
25 </Policy>

17 [0055] The policy is then pruned:

18 <Policy>  
19 <FirstAssertion/>  
20 <SecondAssertion/>  
21 <TrickySecurityAssertion>  
22 <Feature1/>  
23 <Feature2/>  
24 <Feature3/>  
25 </TrickySecurityAssertion>  
<XAlmostLastAssertion>  
<FeatureA>  
<FeatureB>  
</XAlmostLastAssertion>  
<XLastAssertion/>  
</Policy>

[0056] The pruned policy is converted to a string (stream of bytes), for example, using standard security XML QName, white space, namespace, etc., canonicalization. The string is MD5 hashed, and the resulting 128 bit number converted to base 64. The encoded bit vector is then placed in either the fh or mh attribute of the <SelectedPolicy> header.

[0057] The SelectedPolicies SOAP header block may be used to communicate both which policies are applied and how they are applied in the construction of the message. The normative schema outline for the SelectedPolicies header block is:

```
<nsp:SelectedPolicies
  xmlns:nsp="http://schemas.microsoft.com/net/2003/09/policy"
  S:mustUnderstand="xs:boolean"?
  S:role="xs:anyURI"?
  S:relay="xs:boolean"?
  ... >
  <nsp:SelectedPolicy Digest="xs:base64Binary"
    DigestAlgorithm="xs:QName"?
    ... >
    <nsp:SelectedAssertions>xs:base64Binary</nsp:SelectedAssertions>
    ...
  </nsp:SelectedPolicy> +
  ...
</nsp:SelectedPolicies>
```

[0058] If the digest does not correspond to a known policy at the host, or if the cached policy has expired, the host may generate a SOAP Fault:

```
S:Code/S:Value: S:Sender
S:Code/S:Subcode/S:Value: nsp:PolicyUnknown
S:Reason: "Unknown policy"
```

S:Detail/nsp:SelectedPolicy: the child of the SelectedPolicies header block with the digest of the unknown or expired policy.

[0059] The bit vector is computed over the assertions that appear within the policy before including any expressions referenced by an embedded policy reference. If a referenced expression is selected to satisfy the policy, a separate SelectedPolicy child is included in the SelectedPolicies header block to indicate which assertions were selected within that expression. The additional SelectedPolicy children follow the SelectedPolicy child for the referencing policy in the order in which they are referenced.

[0060] This rule may apply whether the referenced expression was referenced by the initial policy or by some referenced expression. For example, if Policy A is selected and references Policy B and then Policy C, and if Policy B references Policy B1 and then B2, then the SelectedPolicies header block would contain five SelectedPolicy children in the following order: A, B, B1, B2, C.

[0061] If the bit vector does not indicate a valid satisfaction of the overall policy, if the bit vector is invalid or incorrectly encoded, or if the message does not satisfy the overall policy in the way indicated by the bit vector, a host generates a SOAP Fault:

S:Code/S:Value: S:Sender

S:Code/S:Subcode/S:Value: nsp:PolicySelectionInvalid

S:Reason: "Invalid set of assertions and policy references selected"

S:Detail/nsp:SelectedPolicy: the child of the SelectedPolicies header block with the incorrect bit vector.

[0062] As another illustrative implementation, a policy digest may be generated as follows. The selected assertions are underlined in the following cached policy (referred to as P).

```
<wsp:Policy
  xmlns:wsp='http://schemas.xmlsoap.org/ws/2002/12/policy'
  xmlns:x='http://example.org' >
  <wsp:OneOrMore>
    <wsp:All>
      <x:A1>abc</x:A1>
      <x:A2 x="123" />
    </wsp:All>
    <x:A3 />
    <x:A4 />
  </wsp:OneOrMore>
  <x:A5 y="123">abc</x:A5>
</wsp:Policy>
```

[0063] Removing all insignificant white space from cached policy P results in the following representation (referred to as P1'):

```
<wsp:Policy xmlns:wsp='http://schemas.xmlsoap.org/ws/2002/12/policy'
xmlns:x='http://example.org'><wsp:OneOrMore><wsp:All><x:A1>abc</x:A1><x:A2
x="123"/></wsp:All><x:A3/><x:A4/></wsp:OneOrMore><x:A5
y="123">abc</x:A5></wsp:Policy>
```

[0064] The policy identity may be computed as follows. The MD5 hash of P1' may be represented in hexadecimal digits as "5faaa79650ae2010a9646174f84ca60a". The base64 encoding represented in text characters is "X6qnllCulBCpZGF0+EymCg==".

[0065] The selected assertions within that policy can be identified as follows. Since there are five assertions in P1 (A1, A2, ..., A5), there is a five-bit vector. The bit vector corresponding to the selected assertions indicated above in binary digits is "11101", or in hexadecimal digits is "1d". The equivalent base64 encoding represented in text characters is "HQ=".

[0066] In another illustrative implementation, a policy digest may be generated for a normalized policy as follows. A policy in normal form has exactly one “OneOrMore” operator whose only children are “All” operators. Each All operator has only children which are assertions. Each of the All operators is sometimes called an “assertion set” (also referred to as a “row”). As an illustration, the normal form of policy P1 (above) is shown in the following cached policy (referred to as NP1).

```
<wsp:Policy
  xmlns:wsp='http://schemas.xmlsoap.org/ws/2002/12/policy'
  xmlns:x='http://example.org' >
  <wsp:OneOrMore>
    <wsp:All>
      <x:A1>abc</x:A1>
      <x:A2 x="123" />
      <x:A3 />
      <x:A5 y="123">abc</x:A5>
    </wsp:All>
    <wsp:All>
      <x:A1>abc</x:A1>
      <x:A2 x="123" />
      <x:A4 />
      <x:A5 y="123">abc</x:A5>
    </wsp:All>
  </wsp:OneOrMore>
</wsp:Policy>
```

[0067] NP1 is semantically equivalent to P1. The selected assertion set in NP1 (illustrated by underlining) is equivalent to the selected assertions in P1 above. Removing all insignificant white space from cached policy NP1 results in the following representation (referred to as AS1'):

```
<wsp:All><x:A1>abc</x:A1><x:A2 x="123"/><x:A3/><x:A5 y="123">abc</x:A5></wsp:All>
```

[0068] The policy digest may then be computed as a row hash. According to such an implementation, the rows still correspond to the rows in the host policy

1 and the policy digest is valid even if the host policy expands. For example, the  
2 MD5 hash of AS1' represented in hexadecimal digits is  
3 "15b157f6ed9d592ba9c83bb5830a948d". The base64 encoding represented in text  
4 characters is "FbFX9u2dWSupyDu1gwqUjQ==".

5 [0069] It is noted that the above implementations are provided for purposes  
6 of illustration. Still other implementations are also contemplated.

### 7 8 Exemplary Operations

9  
10 [0070] Described herein are exemplary methods for implementing invalid  
11 policy detection in a network environment. The methods described herein may be  
12 embodied as logic instructions on one or more computer-readable medium. When  
13 executed on a processor, the logic instructions cause a general purpose computing  
14 device to be programmed as a special-purpose machine that implements the  
15 described methods. In the following exemplary operations, the components and  
16 connections depicted in the figures may be used to implement invalid policy  
17 detection in a network environment.

18  
19 [0071] **Fig. 4** is a flowchart illustrating exemplary operations 400 that may  
20 be implemented to detect an invalid policy. The exemplary operations may be  
21 implemented, e.g., at a client, to “age” one or more policies stored in cache. A  
22 policy may be removed from the cache after a specified time (e.g., a limit value) to  
23 reduce the use of invalid or expired policies. In an exemplary implementation, the  
24  
25

1 limit value may be specified based on how the system is used. For example, if  
2 thousands of messages are being sent per second to a host producing 1% faults, it  
3 is acceptable to have a relatively high limit value specified in the hundreds.  
4 However, in a more reliable system a lower limit value may be specified.

5 [0072] In operation 410, the client may retrieve a policy from cache. For  
6 example, the policy may have been stored in cache during a previous  
7 communication with the host. In operation 420, the policy is evaluated at the client  
8 to determine whether the policy is expired. If the policy is expired, the client  
9 retrieves a current policy from the host in operation 425. In operation 430, the  
10 client applies the policy to the messages it sends to the host.  
11

12 [0073] In operation 440, the client retrieves a counter corresponding to use  
13 of the policy. The counter is evaluated in operation 450. If the counter does not  
14 exceed a limit value, the counter is incremented in operation 455 to indicate that  
15 the policy has been used. Alternatively, if the counter exceeds the limit value in  
16 operation 450, the client removes the policy from cache in operation 460 so that  
17 the policy cannot be used again. A current policy is retrieved in operation 470 for  
18 communications with the host. The counter is reset in operation 480 for the new  
19 policy.  
20

21 [0074] It is noted that the counter can be incremented at any desired rate,  
22 and a policy may expire at any desired count. For example, the counter may be  
23 incremented by 1 count each time the policy is used and the policy may expire  
24  
25



1 when the counter exceeds 300. However, the counter may be incremented by 100  
2 each time the client receives a fault to “age” the policy at a faster rate.

3 [0075] It is also noted that the client can retrieve the current policy from the  
4 host in operation 470 at any suitable time, and the policy does not need to be  
5 retrieved immediately upon removing the expired policy from cache. For example,  
6 the client may retrieve the current policy from the host when other processes are  
7 idle, or during off-peak hours for network traffic. As another example, the client  
8 may retrieve the current policy from the host the next time that the client attempts  
9 to communicate with the host.  
10

11 [0076] **Fig. 5** is a flowchart illustrating exemplary operations 500 that may  
12 be implemented to detect an invalid policy. The operations may be implemented,  
13 e.g., at a client to generate a policy digest of one or more cached policies. The  
14 policy digest may be included in a message to the host so that the host can  
15 determine if the client is using a valid policy.  
16

17 [0077] In operation 510 the client retrieves a policy from cache. For  
18 example, the cached policy may have been written to cache during a previous  
19 communication with the host. In operation 515, the policy is identified. In  
20 operation 520, an assertion is read from the cached policy. In operation 530, the  
21 assertion is assigned a bit value. For example, the bit value may be “1” if the  
22 assertion is selected and “0” if the assertion is not selected. The bit value is written  
23 to a bit vector in operation 540. In operation 550, it is determined whether the  
24  
25

1 policy includes another assertion. If the policy includes another assertion, the  
2 process returns (illustrated by arrow 555) to operations 520-540, where the  
3 assertion is read, assigned a bit value, and written to a bit vector.

4 [0078] If the policy does not include further assertions, the bit vector and  
5 policy identity are written to the policy digest in operation 560. In operation 570,  
6 the policy digest is encoded. The policy digest is written to the message header in  
7 operation 580.

8 [0079] It is noted that the operations described with regard to Fig. 5 may be  
9 applied to multiple policies. For example, a policy digest may be computed for a  
10 group of policies, and bit vectors may be computed for each assertion across all of  
11 the policies. In addition, the operations are not limited to any particular order. For  
12 example, the policy identity and bit vector do not need to be written to the policy  
13 digest at the same time or in any particular order.

14 [0080] **Fig. 6** is a flowchart illustrating exemplary operations 600 that may  
15 be implemented to detect an invalid policy. The operations may be implemented,  
16 e.g., at a host to extract a policy digest from a message. The policy digest may be  
17 evaluated to determine if the client is using a valid policy and which assertions the  
18 client is complying with.

19 [0081] In operation 610 the host extracts a policy digest from a message,  
20 e.g., from the message header. In operation 620, the policy digest is evaluated, e.g.,  
21 to determine the policy identity and the length of the bit vector. If the policy digest  
22  
23  
24  
25

1 is invalid (e.g., incorrect policy identity and/or bit vector length), the host  
2 generates an invalid digest fault in operation 625. In an exemplary implementation,  
3 the fault is identified as an invalid digest fault, indicating that the policy itself is  
4 invalid.

5 [0082] The host may return the invalid digest fault to the client so that the  
6 client can remove the policy from cache. It is noted that when the host determines  
7 that the digest is itself invalid, the host does not need to evaluate the contents of  
8 the policy digest, e.g., for selected references and/or assertions. Accordingly, these  
9 operations save time and reduce the burden on processing resources at the host.  
10 The client is quickly notified that the cache is invalid and can request a current  
11 policy from the host.  
12

13 [0083] If the policy is itself valid, the host may evaluate the selected  
14 assertions in the policy digest. In operation 630 the assertions identified in the  
15 policy digest are compared those in the host policy. In operation 640, a  
16 determination is made whether the client's cached policy is valid, e.g., based on the  
17 selected assertions and/or references contained in the policy digest. If the  
18 assertions are invalid, the host generates an invalid policy fault in operation 645. In  
19 an exemplary implementation, the fault is identified as an invalid policy fault,  
20 indicating that the policy is invalid. The host may return the invalid policy fault to  
21 the client so that the client can remove the policy from cache.  
22  
23  
24  
25

1 [0084] If on the other hand the policy is valid, the host may process the  
2 message in operation 650, and communication can be established between the  
3 client and the host.  
4

### 5 **Exemplary Computing Device**

6 [0085] Fig. 7 is a schematic illustration of an exemplary computing device  
7 700 that can be utilized to implement a host. Computing device 700 includes one  
8 or more processors or processing units 732, a system memory 734, and a bus 736  
9 that couples various system components including the system memory 734 to  
10 processors 732. The bus 736 represents one or more of any of several types of bus  
11 structures, including a memory bus or memory controller, a peripheral bus, an  
12 accelerated graphics port, and a processor or local bus using any of a variety of bus  
13 architectures. The system memory 734 includes read only memory (ROM) 738 and  
14 random access memory (RAM) 740. A basic input/output system (BIOS) 742,  
15 containing the basic routines that help to transfer information between elements  
16 within computing device 700, such as during start-up, is stored in ROM 738.  
17

18 [0086] Computing device 700 further includes a hard disk drive 744 for  
19 reading from and writing to a hard disk (not shown), and may include a magnetic  
20 disk drive 746 for reading from and writing to a removable magnetic disk 748, and  
21 an optical disk drive 750 for reading from or writing to a removable optical disk  
22 752 such as a CD ROM or other optical media. The hard disk drive 744, magnetic  
23  
24  
25

1 disk drive 746, and optical disk drive 750 are connected to the bus 736 by  
2 appropriate interfaces 754a, 754b, and 754c. The drives and their associated  
3 computer-readable media provide nonvolatile storage of computer-readable  
4 instructions, data structures, program modules and other data for computing device  
5 700. Although the exemplary environment described herein employs a hard disk, a  
6 removable magnetic disk 748 and a removable optical disk 752, other types of  
7 computer-readable media such as magnetic cassettes, flash memory cards, digital  
8 video disks, random access memories (RAMs), read only memories (ROMs), and  
9 the like, may also be used in the exemplary operating environment.  
10

11 [0087] A number of program modules may be stored on the hard disk 744,  
12 magnetic disk 748, optical disk 752, ROM 738, or RAM 740, including an  
13 operating system 758, one or more application programs 760, other program  
14 modules 762, and program data 764. A user may enter commands and information  
15 into computing device 700 through input devices such as a keyboard 766 and a  
16 pointing device 768. Other input devices (not shown) may include a microphone,  
17 joystick, game pad, satellite dish, scanner, or the like. These and other input  
18 devices are connected to the processing unit 732 through an interface 756 that is  
19 coupled to the bus 736. A monitor 772 or other type of display device is also  
20 connected to the bus 736 via an interface, such as a video adapter 774.  
21

22 [0088] Generally, the data processors of computing device 700 are  
23 programmed by means of instructions stored at different times in the various  
24  
25

1 computer-readable storage media of the computer. Programs and operating systems  
2 may be distributed, for example, on floppy disks, CD-ROMs, or electronically, and  
3 are installed or loaded into the secondary memory of a computer. At execution, the  
4 programs are loaded at least partially into the computer's primary electronic  
5 memory.

6 [0089] Computing device 700 may operate in a networked environment  
7 using logical connections to one or more remote computers, such as a remote  
8 computer 776. The remote computer 776 may be a personal computer, a server, a  
9 router, a network PC, a peer device or other common network node, and typically  
10 includes many or all of the elements described above relative to computing device  
11 700. The logical connections depicted in Fig. 7 include a LAN 780 and a WAN  
12 782.  
13

14 [0090] When used in a LAN networking environment, computing device  
15 700 is connected to the local network 780 through a network interface or adapter  
16 784. When used in a WAN networking environment, computing device 700  
17 typically includes a modem 786 or other means for establishing communications  
18 over the wide area network 782, such as the Internet. The modem 786, which may  
19 be internal or external, is connected to the bus 736 via a serial port interface 756.  
20 In a networked environment, program modules depicted relative to the computing  
21 device 700, or portions thereof, may be stored in the remote memory storage  
22 device. It will be appreciated that the network connections shown are exemplary  
23  
24  
25

1 and other means of establishing a communications link between the computers  
2 may be used.

3 [0091] Hosts may include host adapter hardware and software to enable a  
4 connection to the communication network. The connection to communication  
5 network may be through an optical coupling or more conventional conductive  
6 cabling depending on the bandwidth requirements. A host adapter may be  
7 implemented as a plug-in card on computing device 700. Hosts may implement  
8 any number of host adapters to provide as many connections to communication  
9 network as the hardware and software support.  
10

11 [0092] In addition to the specific implementations explicitly set forth herein,  
12 other aspects and implementations will be apparent to those skilled in the art from  
13 consideration of the specification disclosed herein. It is intended that the  
14 specification and illustrated implementations be considered as examples only, with  
15 a true scope and spirit of the following claims.  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25